

Projektowanie Systemów Informatycznych

Zajęcia 1

Wprowadzenie

Projektowanie nowoczesnych systemów informatycznych wspierających analizę danych i procesy decyzyjne oparte na algorytmach data science

Rozgrzewka z R

Zajęcia 2

Cykl życia systemu informatycznego – etapy i krótka charakterystyka:

Planowanie (identyfikacja celów), **Analiza wymagań** (określenie funkcji systemu),
Projektowanie (architektura i struktura danych), **Implementacja** (kodowanie),
Testowanie (weryfikacja poprawności), **Wdrożenie** (uruchomienie systemu),
Eksploatacja i utrzymanie (monitorowanie, aktualizacje), **Wycofanie** (zamknięcie systemu).

Data Science workflow

Cykl życia procesu analizy danych w data science:

1. Zdefiniuj cel (Jaki problem staram się rozwiązać?)
2. Zgromadź dane i zarządzaj nimi (Jakie informacje są mi potrzebne?)
3. Zbuduj model (Znajdź w danych wzorce prowadzące do rozwiązań)
4. Oceń model i poddaj go krytyce (Czy model rozwiązuje mój problem?)
5. Zaprezentuj wyniki i udokumentuj je (Udowodnij, że możesz rozwiązać problem i pokaż, jak tego dokonasz)
6. Wdróż model (Wdróż model tak, aby rozwiązywał problem w środowisku produkcyjnym. Wdrażanie i utrzymywanie modelu)

Gromadzenie i analiza wymagań funkcjonalnych i нефункциональных w projektowaniu systemów informatycznych

Analiza wymagań: określenie funkcji systemu

Spisanie tych wymagań pozwala na stworzenie systemu oprogramowania, który spełnia oczekiwania klienta (zamawiającego) w określonych ramach czasowych i budżetowych oraz jest zgodny z celami systemu zidentyfikowanymi w poprzednim etapie (tj. Planowanie).

Wymagania funkcjonalne opisują, CO system ma robić, czyli jakie funkcje ma realizować. Odpowiadają na pytania: "Co system ma umożliwić?", "Jakie zadania ma wykonywać?".

Przykłady wymagań funkcjonalnych:

- Możliwość logowania się użytkowników.
- Wyszukiwanie produktów w sklepie internetowym.
- Generowanie raportów finansowych.

Wymagania нефункциональные opisują, JAK system ma działać, czyli jakie ma mieć cechy jakościowe. Odpowiadają na pytania: "Jak system ma działać?", "Jakie ma mieć właściwości?".

Przykłady wymagań нефункциональных:

- Wydajność (np. czas odpowiedzi systemu).
- Bezpieczeństwo (np. ochrona danych przed nieautoryzowanym dostępem).
- Niezawodność (np. dostępność systemu przez 24/7).
- Użyteczność (np. łatwość obsługi interfejsu użytkownika).

Proces gromadzenia i analizy wymagań:

1. **Identyfikacja interesariuszy:** należy zidentyfikować wszystkie osoby/grupy osób, które są zainteresowane systemem lub będą z niego korzystać.
2. **Gromadzenie wymagań:** stosuje się różne techniki odkrywania wymagań: burze mózgów, konsultacje i wywiady z kluczowymi użytkownikami, analiza dokumentów, prototypowanie.
3. **Analiza wymagań:** należy sprawdzić, czy wymagania są kompletne (niczego nie pominęliśmy), jednoznaczne (bez różnych interpretacji), spójne (niesprzeczne), testowalne (mieralne).
4. **Dokumentowanie wymagań:** wymagania są zapisywane w formie dokumentu specyfikacji wymagań, który jest podstawą do następnego etapu (tj. Projektowanie).
5. **Walidacja wymagań:** należy upewnić się, że zgromadzone wymagania są zgodne z rzeczywistymi oczekiwaniami interesariuszy. Wymagania powinny być regularnie weryfikowane i aktualizowane w trakcie trwania projektu.

Dokumentacja i specyfikacja wymagań w procesie projektowania systemów informatycznych

Dokumentacja wymagań:

- Jest zbiorem dokumentów, które opisują wymagania systemu.
- Służy jako punkt odniesienia dla wszystkich etapów projektu.
- Powinna być zrozumiała, spójna i aktualna.
- Obejmuje m.in.:
 - Specyfikację wymagań oprogramowania (Software Requirements Specification, SRS).
 - Przypadki użycia (use cases).
 - Scenariusze użytkownika (user stories).

Specyfikacja wymagań

- Jest formalnym dokumentem, który szczegółowo opisuje wymagania systemu.
- Stanowi podstawę do następnego etapu (tj. Projektowanie).
- Zazwyczaj zawiera:
 - Opis celów systemu.
 - Opis wymagań funkcjonalnych i нефункциональных.
 - Opis interfejsów użytkownika.
 - Opis wymagań dotyczących danych.

Znaczenie dokumentacji i specyfikacji wymagań:

- Zapewnienie jasności i zrozumienia celów projektu.
- Minimalizacja ryzyka nieporozumień i błędów.
- Ułatwienie komunikacji między interesariuszami.
- Zapewnienie spójności i jakości systemu.
- Ułatwienie testowania i utrzymania systemu.
- Lepsze zarządzanie projektem, ocena jego kosztów, przewidywanie terminów realizacji oraz ocena zwrotu z inwestycji (ROI).

Podjęcie Reproducible Research i jego wpływ na replikację wyników

*Reproducible Research polega na dokumentowaniu kodu, danych i metod
w sposób umożliwiający ich powtórzenie.*

Wspiera transparentność, pozwala na replikację wyników i ułatwia współpracę naukową i biznesową.

Reproducible Research odnosi się do zestawu praktyk, które mają na celu zapewnienie, że wyniki badań — zwłaszcza empirycznych, opartych na danych — mogą być **powtórzone i zweryfikowane przez inne osoby**.

Najważniejszą ideą RR jest to, że każdy wynik powinien być:

- **Sprawdzalny** – osoba trzecia powinna móc odtworzyć wyniki na podstawie dostarczonych danych i kodu.
- **Transparentny** – kod, dane i metodyka analizy muszą być jawne i zrozumiałe.
- **Udokumentowany** – wyniki analizy powinny być zintegrowane z opisem metod i kodem źródłowym oraz dokumentacją i specyfikacją wymagań.

Reproducible Research a replikacja wyników

W kontekście nowoczesnych projektów systemów informatycznych, RR umożliwia:

- **Replikację wyników** – ponowne uruchomienie kodu na tych samych danych powinno dać identyczne wyniki.
- **Walidację modeli** – pozwala niezależnie ocenić skuteczność modeli ML/Text Mining.
- **Audyt** – interesariusze systemu mogą przeanalizować pełen przebieg analizy.

Kluczowe elementy RR w data science i text mining

Element	Znaczenie
Kod źródłowy	Wszystkie skrypty analityczne muszą być czytelne, modularne i udostępnione.
Dane wejściowe	Oryginalne dane (lub link do źródła) muszą być dostępne.
Środowisko pracy	Narzędzia, biblioteki i wersje pakietów powinny być zdefiniowane (np. sessionInfo() w R).
Dokumentacja	Opis założeń, kroków przetwarzania, metryk i sposobu interpretacji wyników.
Raport końcowy	Generowany automatycznie (np. R Markdown, Jupyter Notebook, Quarto).

Narzędzia wspierające podejście RR

- **R Markdown / Jupyter Notebook / Quarto** – integracja tekstu opisu i kodu analitycznego do reprodukcji analiz.
- **Git + GitHub** – wersjonowanie kodu i dokumentacji, współpraca zespołowa.
- **Docker / renv / conda** – replikowalne środowiska pracy.
- **Zenodo / OSF.io** – archiwizacja danych i publikacja kodu.

Podejście RR ma realny wpływ na **nowoczesne systemy informatyczne**:

- **Zwiększa wiarygodność wyników** – zarówno w nauce, jak i w biznesie.
- **Przyspiesza rozwój i debugging** – dzięki jasnej dokumentacji i modularności kodu.
- **Ułatwia współpracę interdyscyplinarną** – wszyscy członkowie zespołu pracują na tej samej wersji projektu.
- **Przygotowuje systemy do audytu i zgodności regulacyjnej** – np. w finansach, medycynie.

Reproducible Research to fundament nowoczesnej nauki i inżynierii danych.

W dobie algorytmizacji procesów decyzyjnych, podejście to staje się istotne dla przejrzystości i trwałości rezultatów pracy zespołów projektowych.

Przykład: projekt text mining w R z wykorzystaniem RR

- Plik zawiera modularny kod:
 - wczytanie danych tekstowych
 - preprocessing tekstu (normalizacja, czyszczenie, stemming, tokenizacja)
 - zliczanie częstości słów
 - wizualizacja
 - eksport wyników
- Wszystko udokumentowane w jednym reprodukowalnym pliku HTML / RMarkdown / Jupyter Notebook.
- Udostępnione na profilu GitHub + link do danych.

Metodyki Agile i Waterfall

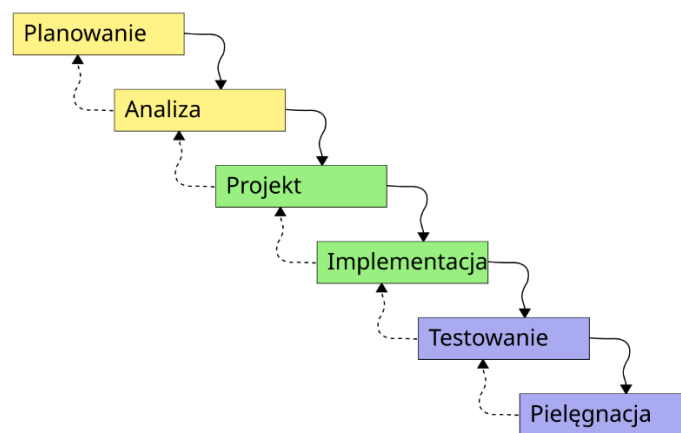
W większości współczesnych procesów wytwarzania oprogramowania, stosuje się jeden z dwóch modeli wytwórczych:

- Model kaskadowy (*waterfall*, wodospadowy)
- Model przyrostowy (*incremental*, iteracyjny); jego rozwinięciem jest model spiralny

Model kaskadowy

https://pl.wikipedia.org/wiki/Model_kaskadowy

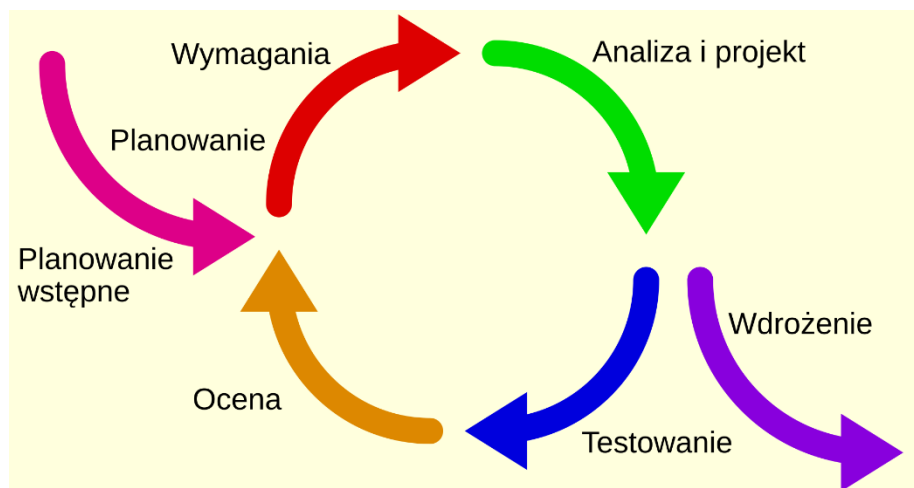
liniowy i sekwencyjny - nie można przejść do następnej fazy przed zakończeniem poprzedniej; błąd popełniony w początkowej fazie ma wpływ na całość; łatwy w nadzorowaniu.



Model przyrostowy

https://pl.wikipedia.org/wiki/Model_przyrostowy

ewolucyjny - pozwala na powroty do faz poprzedzających; umożliwia adaptowanie systemu do zmian w wymaganiach; umożliwia korygowanie popełnionych błędów; trudny w nadzorowaniu.



W praktyce codziennej pracy zespoły projektowe/deweloperskie potrzebują konkretnych wytycznych „jak należy wytworzyć oprogramowanie”. Zespół musi wiedzieć, co należy wykonać, w jaki sposób oraz kto powinien za co odpowiadać, dlatego:

**Modele wytwórcze mają swoje warianty
charakterystyczne dla
odpowiednich metodyk wytwarzania oprogramowania.**

Metodyka to ustandaryzowane podejście do rozwiązywania problemów dla wybranego obszaru.

Metodyki kaskadowe Waterfall – tradycyjne

Wytwarzanie oprogramowania w tradycyjnym podejściu opiera się w uproszczeniu na trzech krokach: zleceniobiorca ustala z klientem co ma zostać zrobione, zespół deweloperski wykonuje projekt, system w całości zostaje przekazany do klienta.

Tutaj każda kolejna faza następuje po kolei jedna za drugą i nie pomija się żadnego etapu. Ewentualne zmiany „w fazie poprzedniej” są trudne do wprowadzenia po rozpoczęciu kolejnej fazy.

Metodyki zwinne Agile – nowoczesne

Sednem metodyk zwinnych Agile jest cykliczność postępu prac nad projektem i oddawanie go klientowi w mniejszych „porcjach”, przyrostowo. Klient widzi na każdym etapie w jakich kierunkach rozwija się projekt i włącza się w proces jego wytwarzania, przekazując swoje uwagi i komentarze (feedback). Następnie te uwagi znów są wysłuchane, analizowane i uwzględnione – cykl powtarza się.

Manifest programowania zwinnego

<https://agilemanifesto.org/iso/pl/manifesto.html>

- Osoby i interakcje ponad procesy i narzędzia
- Działające oprogramowanie ponad kompleksową dokumentację
- Współpraca z klientem ponad negocjacje kontraktowe
- Reagowanie na zmiany ponad sztywne przestrzeganie planu

Podejście zwinne obejmuje co najmniej kilkanaście frameworków i metodyk zwinnych oraz kilkadziesiąt najpopularniejszych praktyk, wiele technik i różnorodnych narzędzi, m.in.:

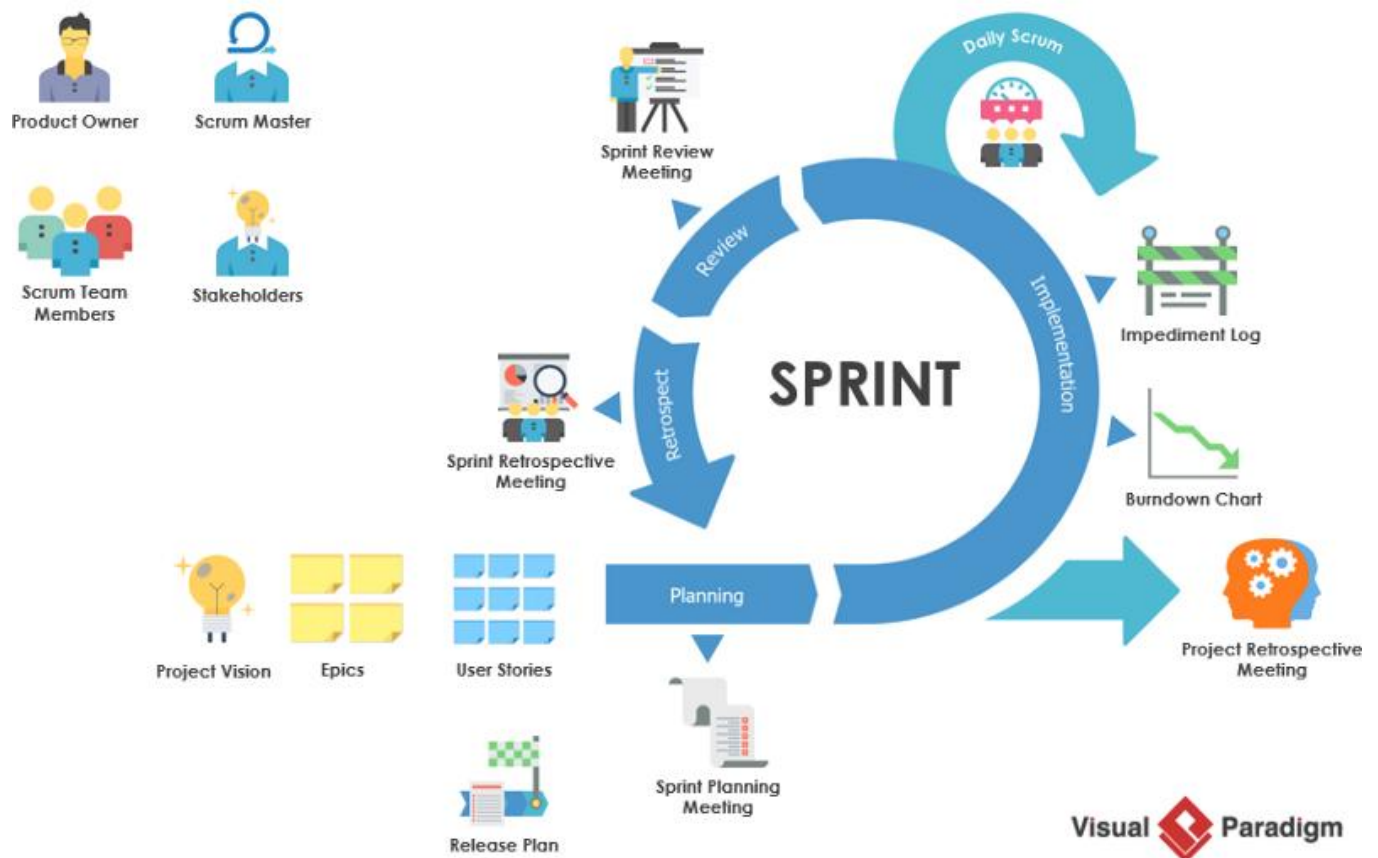
- SCRUM
- KANBAN
- Lean Software Development
- Test-Driven Development (TDD)

Podstawowe różnice pomiędzy podejściem zwinnym wytwarzania oprogramowania i tradycyjnym

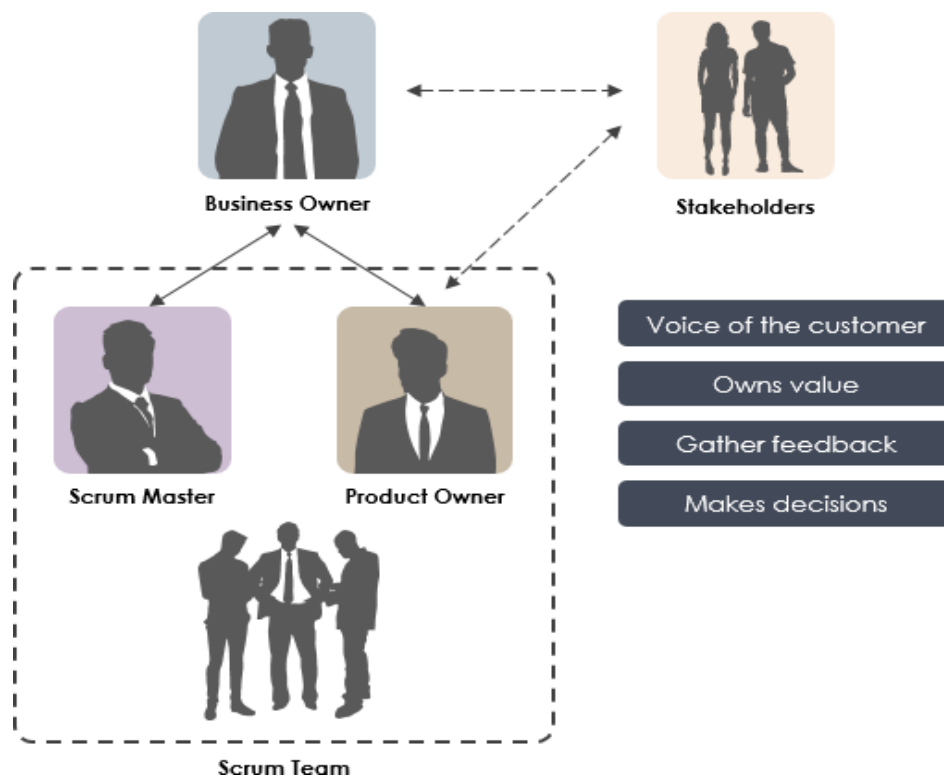
ZWINNE	TRADYCYJNE
Rezultatem jest oprogramowanie, którego klient oczekuje	Rezultatem jest oprogramowanie opisane w dokumentacji projektowej
Projekt podzielony na krótkie etapy	Projekt podzielony na długie etapy
Procesy, zadania i czynności	Wyniki biznesowe, działający produkt
Planowanie i harmonogramowanie	Interakcje i zarządzanie wiedzą
Szybkie efekty w porcjach, stopniowe oddawanie przyrostów	Ostateczny efekt na koniec projektu, cały system wdrażany jednorazowo
Łatwe wprowadzanie korekt i zmian	Wprowadzanie korekt utrudnione
Niższe koszty rezygnacji z projektu	Wysokie koszty rezygnacji z projektu
Mniejsze ryzyko niepowodzenia	Większe ryzyko niepowodzenia
Osiągnięcie określonych korzyści biznesowych	Opracowanie systemu o zdefiniowanych cechach

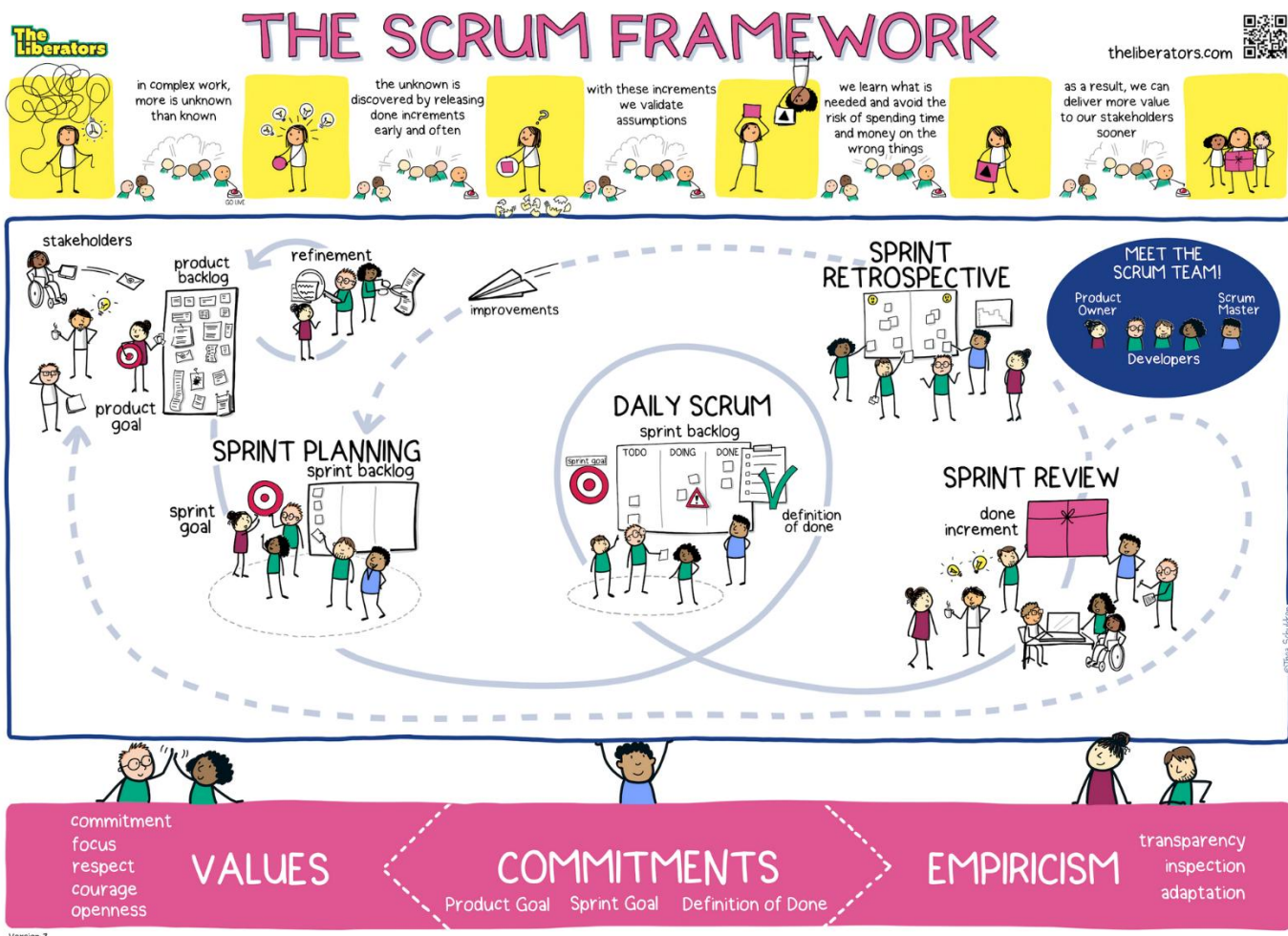
SCRUM framework

The Agile – Scrum Framework



Źródło: <https://www.cybermedian.com/pl/scrum-a-quick-introduction/>





Źródło:

https://shop.theiberators.com/cdn/shop/products/Poster-ScrumFramework-SMALL_1024x1024@2x.png?v=1678871356

Więcej:

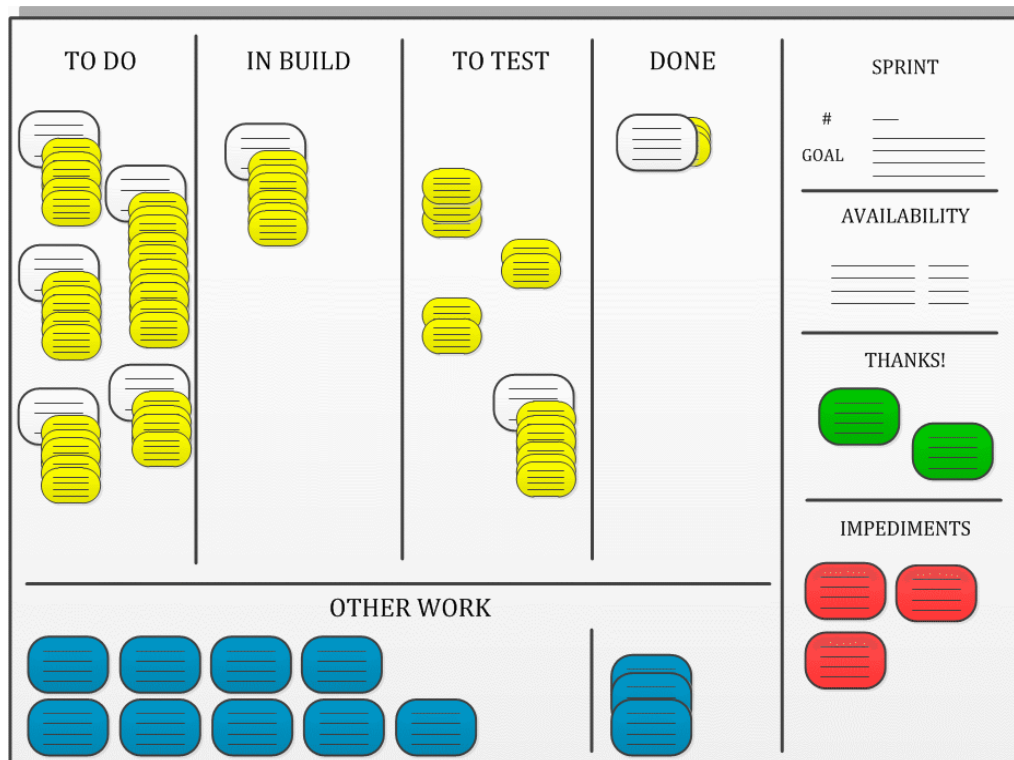
Czym jest SCRUM

<https://procognita.pl/scrum/>

Co to jest SCRUM

<https://agileforce.pl/blog/co-to-jest-scrum/>

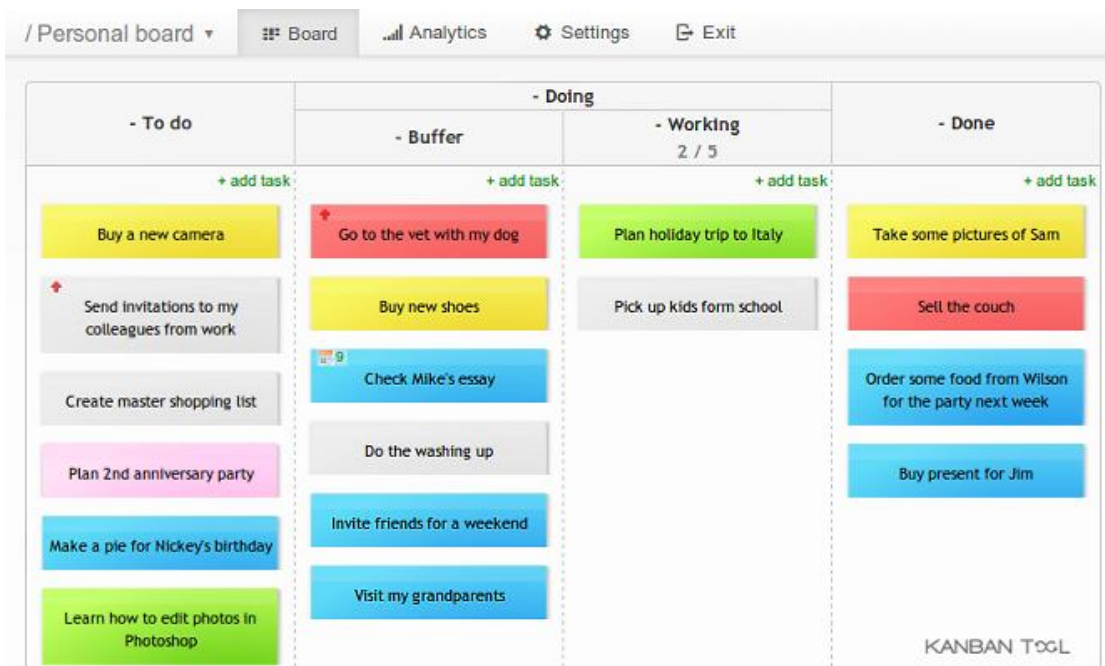
KANBAN to wizualny system zarządzania projektem, który pozwala na efektywną organizację pracy zespołowej i optymalizację przepływu pracy.



Przykładowy product backlog w Trello:

<https://trello.com/b/BLplifUB/datacamp-course-roadmap>

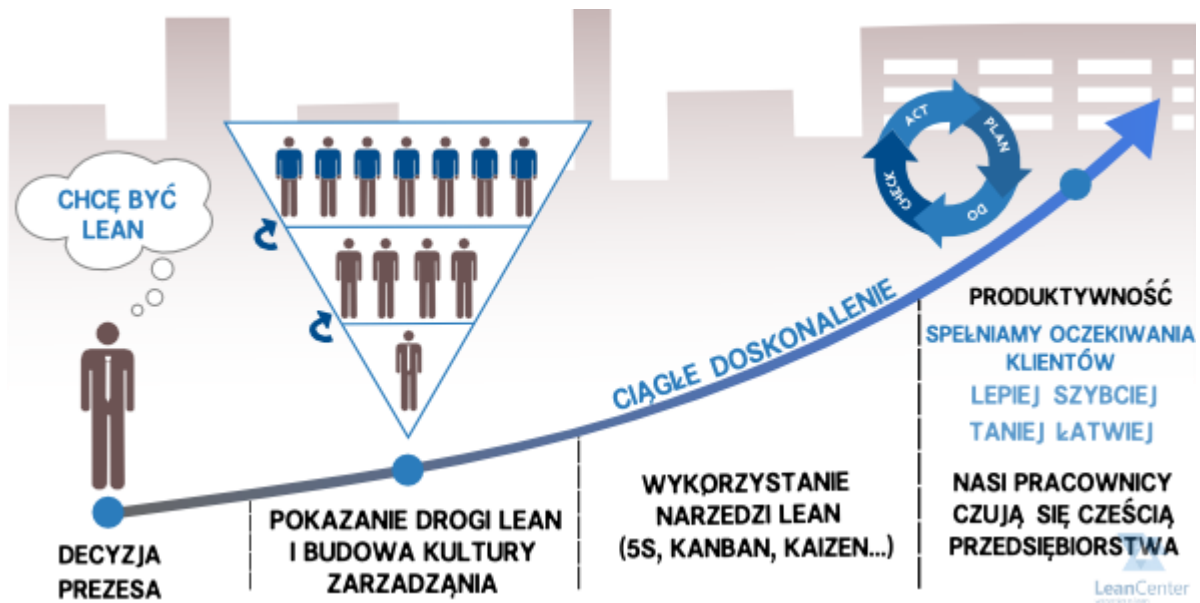
Przykład Osobistej Tablicy Kanban:



Źródło: <https://kanbantool.com/pl/osobista-tablica-kanban>

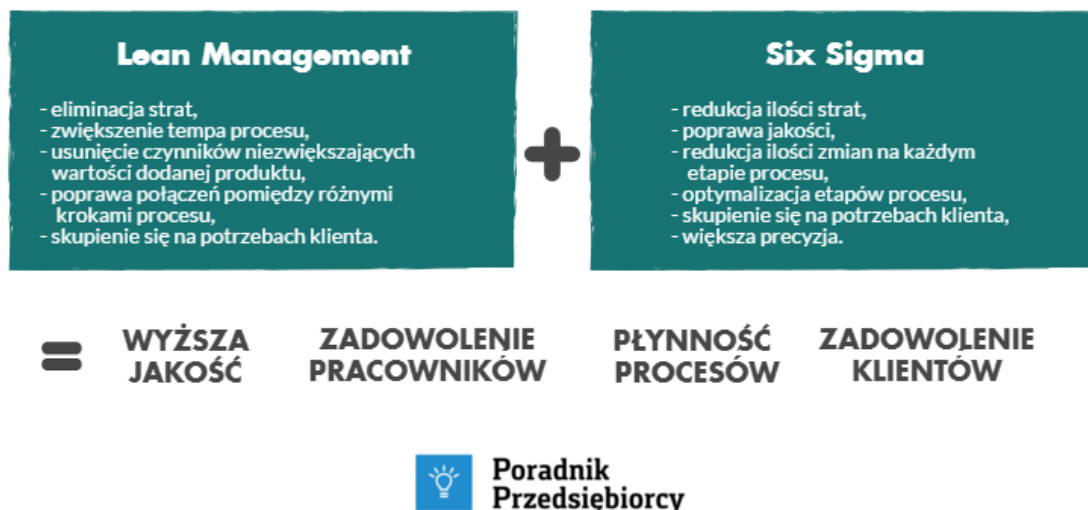
Lean Software Development

Wywodzi się z Lean Management:



Źródło: <https://leancenter.pl/bazawiedzy/lean-management>

Lean Management i Six Sigma - giganci zarządzania razem?

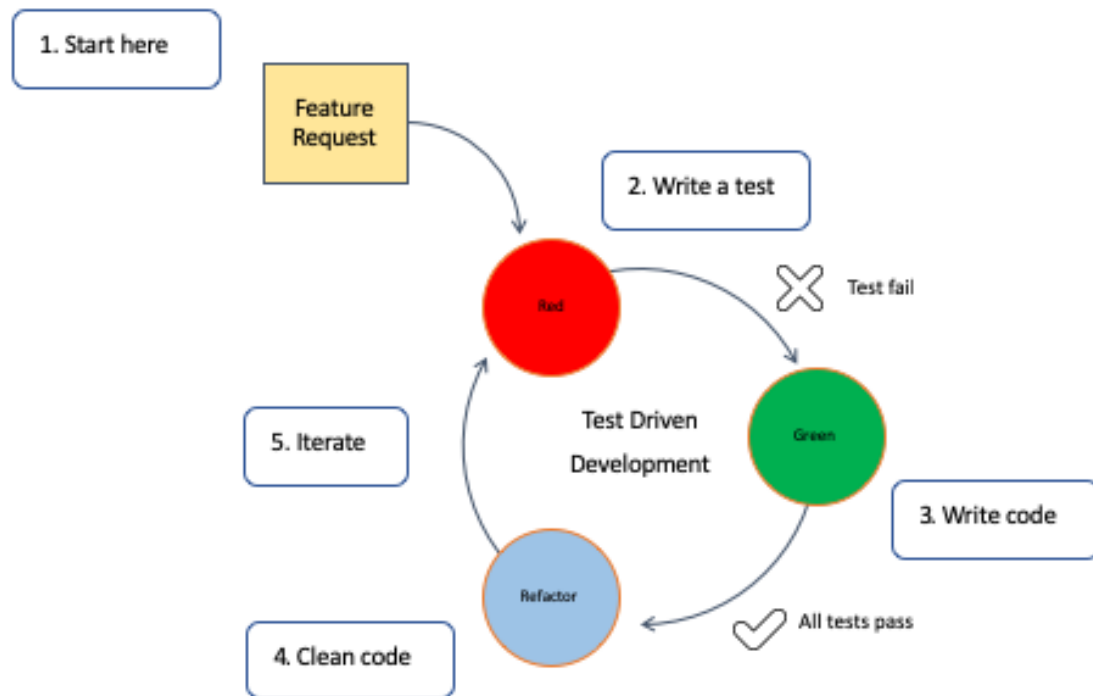


Źródło:

<https://poradnikprzedsiębiorcy.pl/-lean-management-i-six-sigma-nowoczesne-metody-zarzadzania-firma>

Test-Driven Development (TDD)

Test Driven Development to metoda tworzenia oprogramowania, w której pisze się testy przed napisaniem kodu. Założenie jest takie, że najpierw określa się, jak ma działać system, pisząc testy, które to potwierdzą, a następnie implementuje się kod, który będzie spełniał te testy.



Ten przepływ pracy jest czasami nazywany Red-Green-Refactoring, co pochodzi od statusu testów w cyklu:

- Czerwona faza oznacza, że kod nie działa.
- Zielona faza oznacza, że wszystko działa, ale niekoniecznie w najbardziej optymalny sposób.
- Niebieska faza oznacza, że tester refaktoryzuje kod (optymalizuje go), ale też jest pewien, że jego kod jest pokryty testami, co daje mu pewność, że może zmienić i ulepszyć napisany kod.

Źródło:

<https://developer.ibm.com/articles/5-steps-of-test-driven-development/>

Analiza ryzyka w projektowaniu systemów informatycznych. Macierz ryzyka i jej kluczowe elementy

Analiza ryzyka to proces identyfikowania, oceny i priorytetyzacji potencjalnych zagrożeń, które mogą wpłynąć negatywnie na projekt informatyczny – na jego budżet, harmonogram, jakość, bezpieczeństwo lub funkcjonalność.

- Umożliwia **wczesne wykrycie problemów**, zanim wpłyną na system.
- Pozwala **zaplanować działania zapobiegawcze i naprawcze**.
- Poprawia jakość decyzji projektowych i **zwiększa szansę projektu na sukces**.

Główne etapy analizy ryzyka:

1. **Identyfikacja ryzyk**
→ np. opóźnienia, brak zasobów, niestabilne wymagania, awarie technologii, błędy w kodzie.
2. **Szacowanie ryzyka**
→ określenie **prawdopodobieństwa** wystąpienia i **skutków** dla projektu.
3. **Priorytetyzacja**
→ często stosuje się macierz ryzyka (skala: niskie / średnie / wysokie).
4. **Zarządzanie ryzykiem (reakcja na ryzyko)**
 - **Unikanie** (zmiana planu)
 - **Minimalizacja** (redukcja skutków lub prawdopodobieństwa)
 - **Przeniesienie** (np. ubezpieczenie, outsourcing)
 - **Akceptacja** (jeśli ryzyko jest małe lub kosztowne do ograniczenia)
5. **Monitorowanie**
→ regularna aktualizacja listy ryzyk i planów reakcji.

Przykładowe obszary ryzyka w projektowaniu systemów:

- **Techniczne:** błędne założenia, nowa technologia, problemy z integracją.
- **Organizacyjne:** brak decyzyjności, niedobór zasobów, zmiany kadrowe.
- **Użytkownicy:** brak akceptacji, błędna analiza potrzeb.
- **Zewnętrzne:** zmiany przepisów, inflacja kosztów, zależność od dostawców.

Rysunek 1. Przykładowa macierz ryzyka

		PRAWDOPODOBIENSTWO		
		Niskie (1)	Średnie (2)	Wysokie (3)
WPŁYW	Wysoki (3)	ŚREDNIE 3	WYSOKIE 6	BARDZO WYSOKIE 9
	Średni (2)	NISKIE 2	ŚREDNIE 4	WYSOKIE 6
	Niski (1)	BARDZO NISKIE 1	NISKIE 2	ŚREDNIE 3
RYZYO = WPŁYW X NIEBEZPIECZEŃSTWO				

Źródło: <https://www.ifirma.pl/blog/analiza-ryzyka-czym-jest-metody-analazy-ryzyka/>

			SKUTEK				
			Bardzo niski	Niski	Średni	Wysoki	Bardzo wysoki
			1	2	3	4	5
PRAWDOPODOBIENSTWO	Prawie pewne	5	Ś	W	K	K	K
	Prawdopodobne	4	Ś	W	W	K	K
	Możliwe	3	N	Ś	W	W	K
	Mało prawdopodobne	2	N	Ś	Ś	W	W
	Rzadkie	1	N	N	Ś	W	W

Poziom ryzyka	Opis działania
Niski (N)	Poziom ryzyka akceptowany – działania podejmowane w zależności od wymaganych nakładów
Średni (Ś)	Poziom ryzyka nieakceptowany – działanie może zostać przesunięte w czasie, ale wymaga okresowego monitorowania
Wysoki (W)	Poziom ryzyka nieakceptowany – działanie może zostać przesunięte w czasie, ale wymaga stałego monitorowania
Krytyczny (K)	Poziom ryzyka nietolerowany – wymaga natychmiastowego działania

Tabela 3. Przykładowa macierz ryzyka

Źródło: <https://blog-daneosobowe.pl/ocena-ryzyka-w-rodo-jak-sie-za-to-zabrac-cz-2/>

Bibliografia

- Alston J. M., Rich J. A., A Beginner's Guide to Conducting Reproducible Research, 2021, https://www.researchgate.net/publication/348543052_A_Beginner's_Guide_to_Conducting_Reproducible_Research#fullTextFileContent
- Farley D., Nowoczesna inżynieria oprogramowania. Stosowanie skutecznych technik szybszego rozwoju oprogramowania wyższej jakości, Helion 2023
- Hoover D. H., Oshineye A., Praktyka czyni mistrza. Wzorce, inspiracje i praktyki rzemieślników programowania, Helion 2017
- Madeyski L., Kitchenham B., Reproducible Research – What, Why and How, 2017, <https://madeyski.e-informatyka.pl/download/MadeyskiKitchenham15.pdf>
- Przewodnik po Scrumie — czym jest Scrum, jak działa i jak zacząć, <https://www.atlassian.com/pl/agile/scrum>
- Rehkopf M., Porównanie Agile i Scrum. Jak wybrać najlepszą metodologię dla siebie, <https://www.atlassian.com/pl/agile/scrum/agile-vs-scrum>
- Silge J., Robinson D., Text Mining with R: A Tidy Approach, O'Reilly 2024, <https://www.tidytextmining.com/>
- Śmiałek M., Rybiński K., Inżynieria oprogramowania w praktyce. Od wymagań do kodu z językiem UML, Helion 2024
- Wrycza S., Maślankowski J., Informatyka ekonomiczna. Teoria i zastosowania, PWN 2019
- Zumel N., Mount J., Język R i analiza danych w praktyce, Helion 2021